Mina Gao and Justin Sim (Lab Group 4)
EE 371
January 23, 2024
Lab 2 Report

Procedure

This lab consists of three tasks, each related to the memory unit of a DE1_SoC. For all tasks, the read/write/address parameters/inputs were displayed on the board's HEXs. The first task involved manually creating a memory unit. In the second task we utilized the boards built-in ram units to perform a classic read/write operation, as well as a function which counts up from the initial to the final address. For the third task, we built a FIFO memory module that can store/eject data when written/read by keeping track of two memory addresses (read address & write address).

Task #1

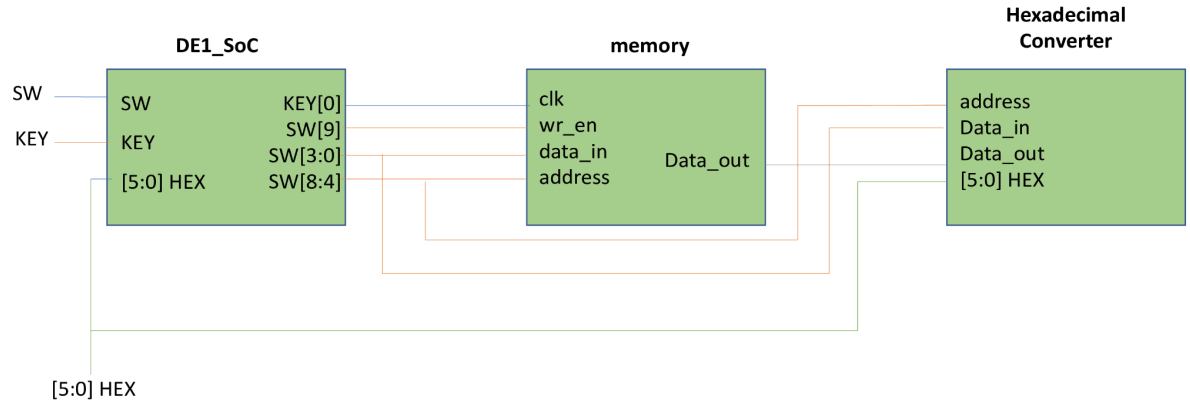

Figure 1: Task 1 Block Diagram

In this task, I created a new Quartus SystemVerilog file to simulate the behavior of a memory unit. I used switches SW 3–0 to input data for the RAM and switches SW 8–4 to specify the address. Switch SW 9 acted as the Write signal, and KEY 0 served as the Clock input. The 7-segment displays HEX5–4 showed the address value in hexadecimal, HEX2 displayed the input data to the memory, and HEX0 showed the data read out of the memory. Additionally, I created testbenches for all modules. After compiling the circuit, I uploaded the design onto the LabsLand DE1_SoC board. To verify the functionality, I applied inputs and observed the output, recording a video to demonstrate the design.

Some difficulties encountered during the process of creating the module was, for one, when using KEY[0] as the clock–we are used to using the in-built CLOCK_50 on the DE1_SoC–and it was difficult to demonstrate the behavior of our RAM on hardware since we had to account for a manual clock input. Another difficulty was in representing the Hexadecimal in addition to the 32 address count. I first tried using the same module for this, but realized that each takes in a different bit length input. Therefore, we made two separate modules to represent the hexadecimal and the 32 addresses.

Task #2

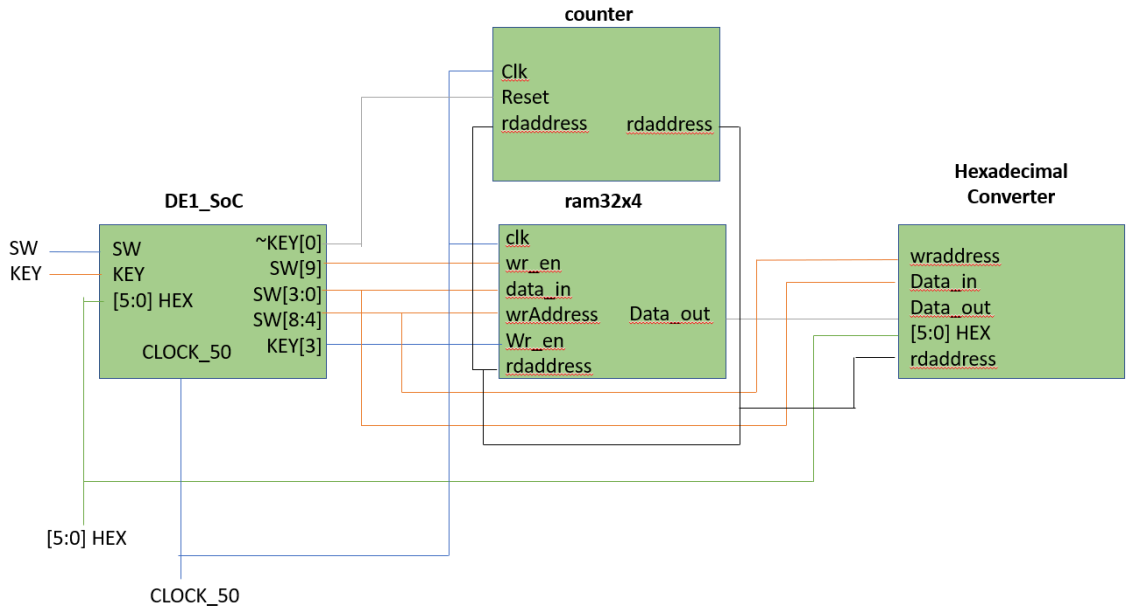

Figure 2: Task 2 Block Diagram

To create a MIF, I followed these steps: Firstly, I chose File -> New -> Memory Initialization file. Then, I specified the number of words as 32 and the word size as 4. Under the View Tab, I adjusted the number of cells per row, along with the address and memory radix. I manually filled the grid with values for each memory address. Afterward, I wrote a SystemVerilog module that instantiated my dual-port memory. I created testbenches for all modules and simulated them before uploading the design onto LabsLand.

To display the RAM contents, I added a capability to my design to show the content of each four-bit word in hexadecimal format on the 7-segment display HEX0–similar to Task 1. Using a counter as a read address, I scrolled through the memory locations, displaying each word for about one second. As each word was displayed, its address in hex format was shown on the 7-segment displays HEX3–2. I utilized the 50 MHz clock, CLOCK_50, with KEY 0 as a reset input, and KEY 3 as my wr_en. For the write address and corresponding data, I used

switches SW 8–4 and SW 3–0, displaying the write address on HEX5–4 and the write data on HEX1.

Some difficulties encountered when working on this task arose in the clock divider portion. Before applying a clock divider, we noticed that our hardware would show a blur on some of the HEXs, which contradicted our modelslim output. This portion was important because it allowed a way to slow down the clock only for the read address i/o to determine the values at a slower rate, about one second at a time.

Task #3

With the provided skeleton project code, I was able to complete the FIFO_Contorl module using the FSM diagram shown below. In order to keep track of the read address and write address, a pointer module is used to archive that. So when a write or read signal is given, the pointer module will return the next writeAddr or readAddr. Then I choose to implement the dual-port RAM using the IP Catalog, following the similar steps shown in Task2. To create a MIF, I followed these steps: Firstly, I chose File -> New -> Memory Initialization file. Then, I specified the number of words as 16 and the word size as 8. Afterward, the FIFO module is used to instantiate my dual-port memory. I created testbenches for all modules and simulated them before uploading the design onto LabsLand.

To demonstrate the design, I utilized the 50 MHz clock, CLOCK_50, with KEY 0 as a reset input, KEY 2 as my write input and KEY 3 as my read input. SW7-SW0 are used to represent the data input and HEX5-HEX4 are used to display the value of the data input in hexadecimal. HEX1-HEX0 are used to show the current data output in hexadecimal. To represent the state of FIFO, LEDR 9 is used to show ‘full’ and LEDR 8 is used to show ‘empty’.

Some difficulties encountered when trying to implement the FSM in FIFO_Contorl. After getting the help from the lab section, we were able to pull out the design and get the simulation to run. Later on, while we tested on the labsland we found we needed a module to buffer and delay the input and we were able to achieve that by using a double flop design in which two flip-flops are used in series to stabilize the signal..

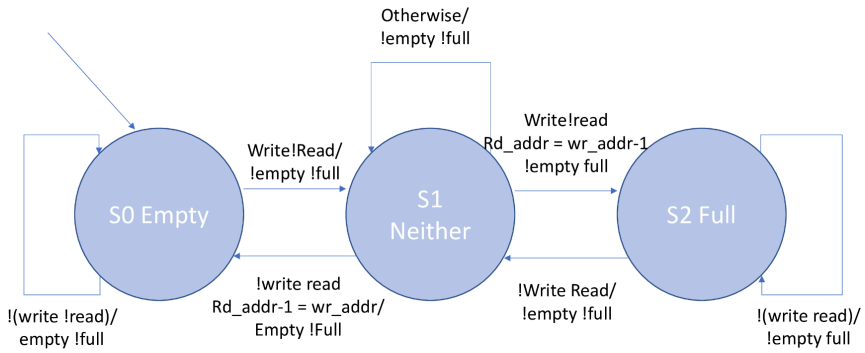

Figure 3: Task 3 Block Diagram

Results

Task 1: DE1_SoC_testbench():

Inputs:

1) SW[9] enable write – highlighted in Pink
2) SW[8:4] input address – highlighted in Blue
3) SW[3:0] input data – highlighted in Yellow
4) KEY[0] - clock

Outputs:

1) HEX[5:4] – address value (HEX5 is first digit)
2) HEX 2 – data being written to memory
3) HEX0 – read memory out

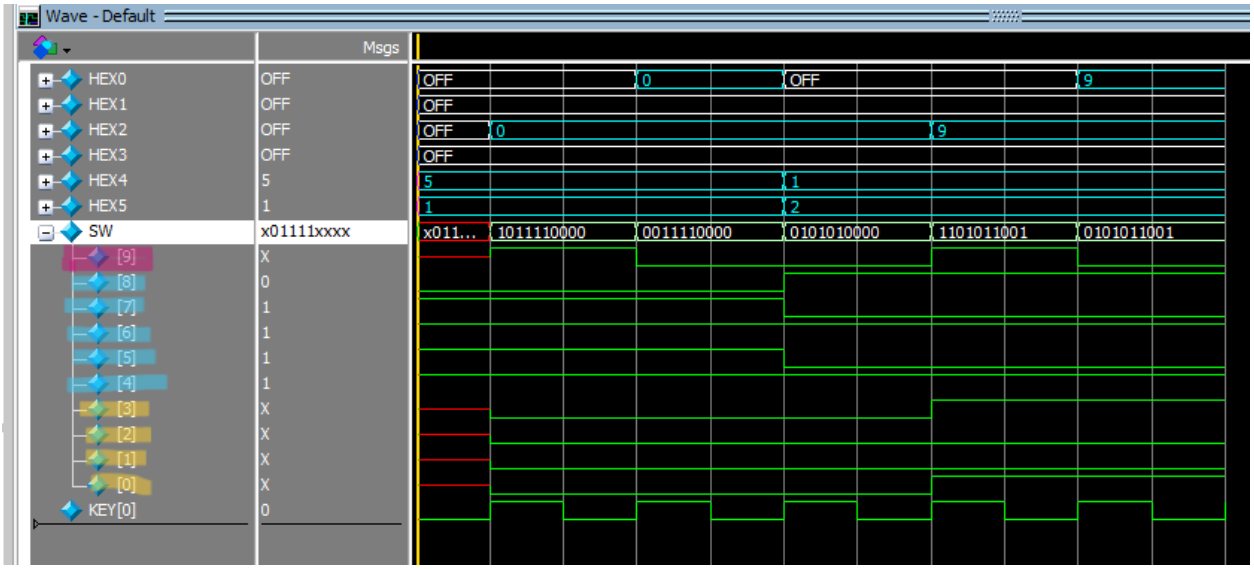

Figure 4: Task 1 DE1_SoC testbench ModelSim

The image above shows a testbench for the module. The intended flow was as such:

1) Display address 15
2) Write “0” to Address 15
3) Read Address 21
4) Write “9” to address 21
5) Read Address 21

According to the figure, it is easy to see that the HEX are following the behaviors intended by our testbench code. One important thing to note is that HEX5 is the first digit because of the way that the board is orientated. Another design choice was for all other non-used HEX to be turned off.

Task 1: M10K_23_4_testbench():

This module behaves similarly to a RAM/memory unit of a DE1_SoC. It was manually programmed to take 3 inputs that can write/store and read to a specific location of the memory unit.

Inputs:

1) Clk
2) Addr
3) Din
4) write

Outputs:

1) dout

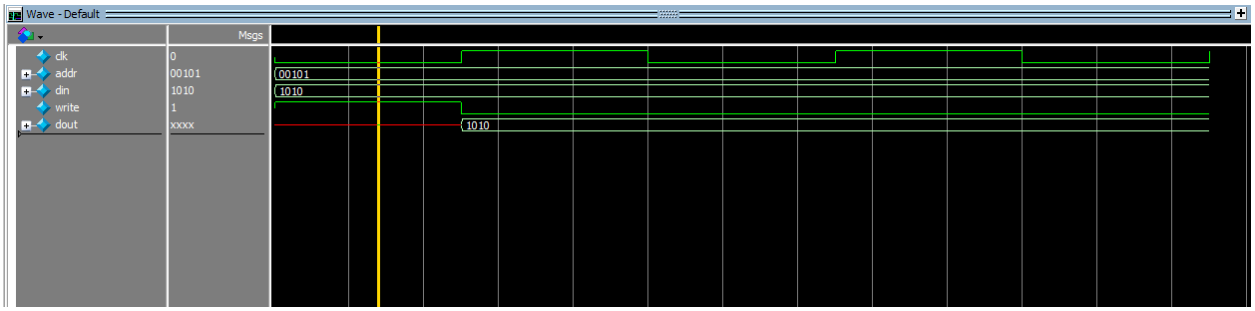

Figure 5: M10K_32_4 testbench ModelSim

The image above shows a testbench for the module. In our testbench code, we wanted to simulate both write and read instances. The expected behavior was as such:

1) Write 1010 to address 00101
2) Read address 00101

We can tell that the module shows successful validation, because the final output is 1010 for when it is reading address 00101, and there were no initial values in the addresses.

Task 2: DE1_SoC testbench():

Inputs:

1) SW[9] enable write
2) SW[8:4] input address
3) SW[3:0] input data
4) KEY[0] - clock

Outputs:

1) HEX[5:4] – address value (HEX5 is first digit)
2) HEX 2 – data being written to memory
3) HEX0 – read memory out

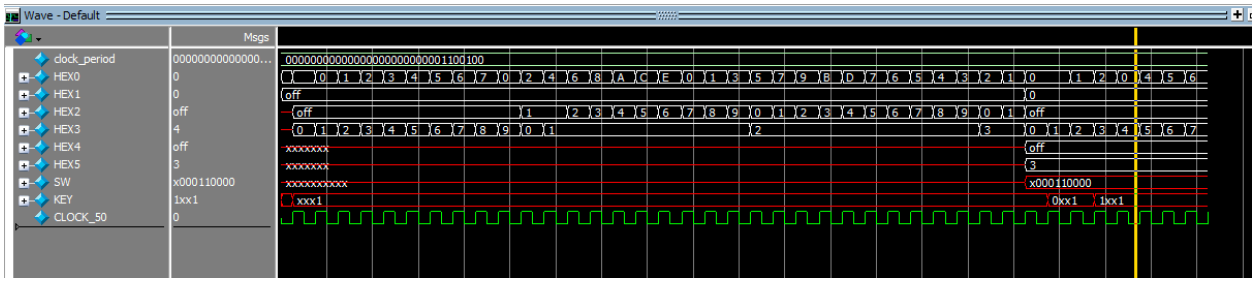

Figure 6: DE1_SoC testbench for Task 2

The image above shows a testbench for the module. We can verify that it shows the correct behavior by seeing if it starts to count up from 0 to 31 on HEX[3:2], which is shown above. Additionally, the testbench image demonstrates a user input of 0 onto address 4 with the addition of a write enable. The resulting output can be seen at the yellow line. These example showed us that the model was ready for hardware implementation.

display_31digit_testbench():

Depending on the 5-bit input, the module will return a number from 0 - 31 in Hexadecimal format. The strategy to returning numbers beyond 0-9 is to pass a second adjacent HEX input such that it acts as the second digit such that each HEX can reach from 0 to 9. It was useful for Tasks 1 and 2 where we were working with 32 addresses and needed a way to identify each of them.

Inputs

1) clk
2) [4:0] count

Outputs:

1) [6:0] HEX0 (first digit)
2) [6:0] HEX1 (second digit

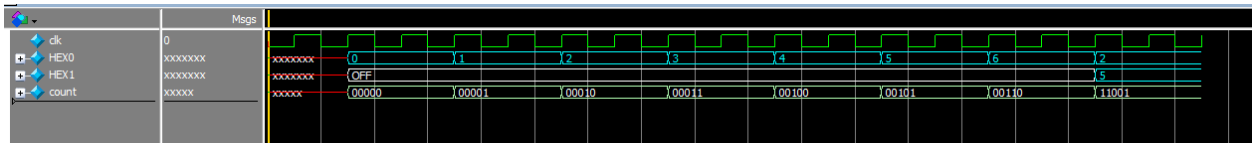

Figure 7: display_31digit testbench Modelsim

The image above shows a testbench for the module. The intended behavior is to display the selected numbers (0,1,2,3,4,5,6,25) for two clock cycles each. It is easy to see that the testbench is aligned with our expectations, meaning that this module can be implemented to perform bit digit to HEX-digit-representation conversion.

HexadecimalConverter_testbench():

Depending on the 4-bit input, it will return the corresponding Hexadecimal character, which was helpful for Task 1, 2, and 3 where we wanted to convert binary input to something readable on the HEX display.

Inputs

1) clk
2) [3:0] binary_input

It has 2 input:

1) [6:0] hexadecimal_output

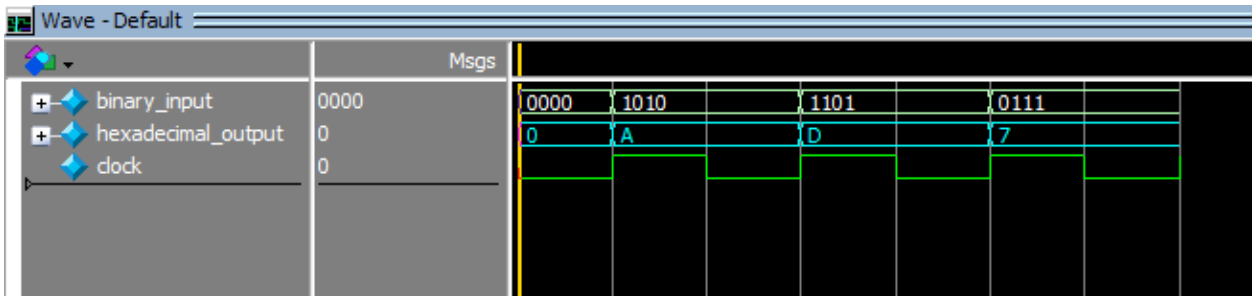

Figure 8: HexadecimalConverter testbench Modelsim

The image above shows a testbench for the module. A radix setting was set following the structure of a common 7-bit hexadecimal conversion (0 to F). The intended behavior is to display the selected characters (A,D,7) for one clock cycle each. It is easy to see that the testbench is aligned with our expectations, meaning that this module can be implemented to perform 4-bit digit to 7-bit hexadecimal conversion.

counter_testbench():

Counter.sv was useful in Task 2 as the primary function for automatically scrolling through addresses. It will increment the input by one bit and return the result. When reset, it resets the input to 0.

Inputs:

1) Clk
2) reset

Outputs:

1) [4:0] rdaddress – read address

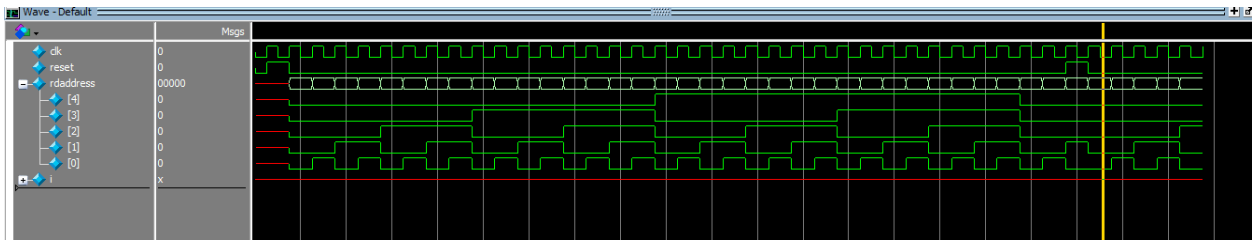

Figure 9: counter testbench Modelsim

The image above shows a testbench for the module. We are able to see that the number represented by 4'b rdadress is incrementing by one for every clock cycle. When the reset is high, the rdaddress returns 4'b0000 (reset location is identified with the yellow bar in figure above). This is aligned with our expectations for the counter module.

clock_divider_testbench():

clock_divider.sv was useful in Task 2 because it allowed for a longer clock cycle. It slows down the clock, so that we were able to scroll through the addresses at a reasonable rate (~1sec)

1) clk
2) reset

Outputs:

1) [31:0] divided_clocks

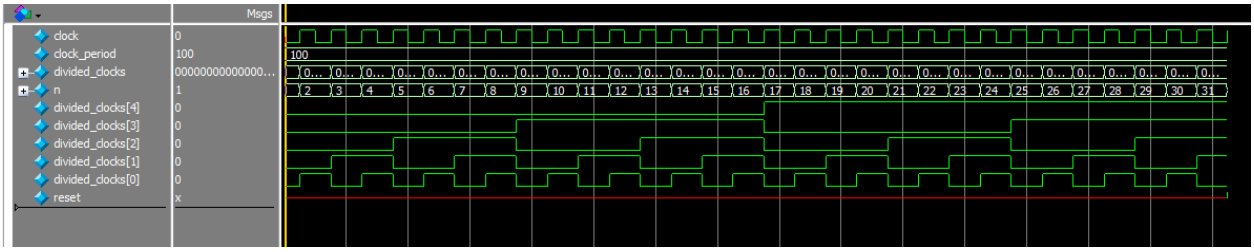

Figure 10: clock_divider testbench Modelsim

The image above shows a testbench for the module. The clock divider has 31 indices, each with a different clock frequency. One can see that divided_clock[1] has double the period of [0], [2] has double the period of [1] and so on. This is consistent with our expectations for the functionality, and it serves a great purpose in customizing the speed of our clock.

Task 3: DE1_SoC_testbench():

Inputs:

1) CLOCK_50
2) reset (KEY[0])
3) read (KEY[3])
3) write (KEY[2])
4) [7:0] inputBus (SW0-7)

Outputs:

1) empty (LEDR 8)
2) full (LEDR 9)
3) [7:0] value of outputBus (HEX1-0)
4) [7:0] value of inputBus (HEX5-4)

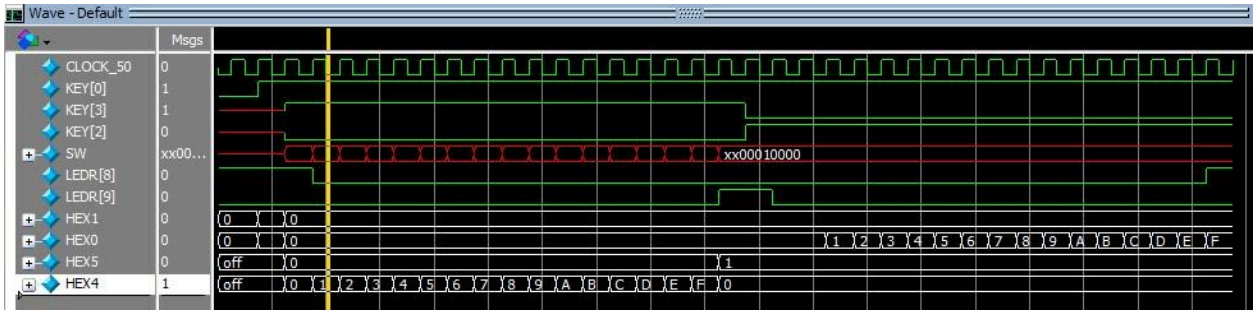

Figure 11: DE1_SoC testbench Modelsim for Task 3

The image above shows a testbench for the module. We can verify that it shows the correct behavior by seeing if the value shown in HEX5-4 matches the data input (SW7-0), and FIFO can keep track of the input data up to 16 places when KEY[2] is pressed. Moreover if the HEX1-0 correctly outputs the written value in order when KEY[3] is pressed; if the LEDR 8 lights up when empty and LEDR 9 lights up when full. The testbench is aligned with our expectations, meaning that this module can be implemented on the DE1_SoC board.

pointer_testbench():

Inputs:

1) clk
2) reset
3) RorW (Read or Write)

Outputs:

1) [3:0] ptr_PorW

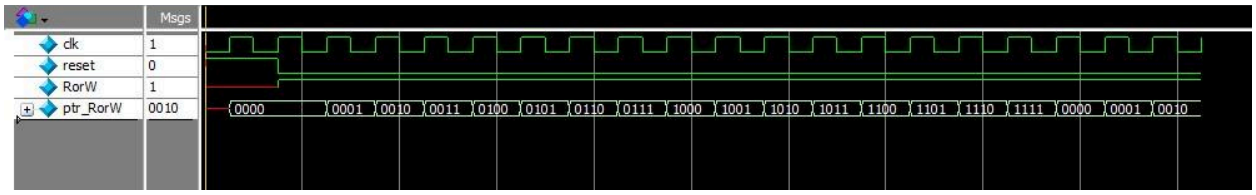

Figure 12: Pointer testbench Modelsim for Task 3

The image above shows a testbench for the module. We can verify that when there is a read or write input, the address of the pointer increases by one at every rising clock edge. Moverover, it will not exceed the upper limit 16, instead it goes back to 0 again.

FIFO_testbench():

Inputs:

1) clk
2) reset
3) read

4) write
5) [7:0] inputBus

Outputs:

1) empty
2) full
3) [7:0] outputBus

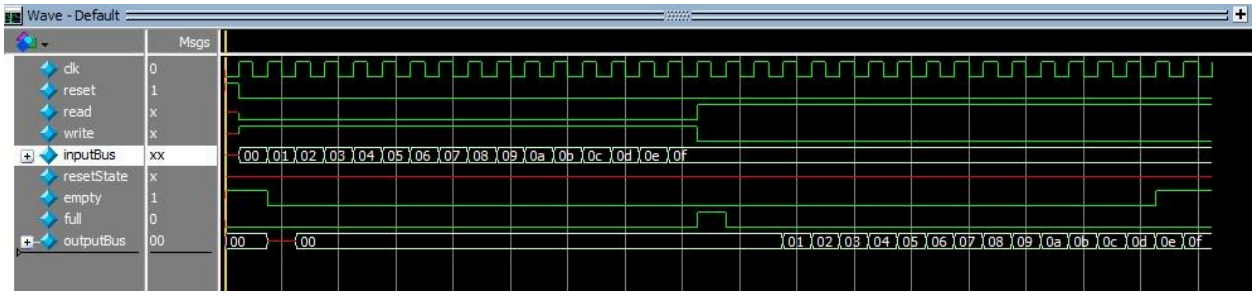

Figure 14: FIFO testbench Modelsim

The image above shows a testbench for the FIFO module. The testing logic for this module is very similar to the FIFO_control module, and the only difference is we want to be able to check if FIFO can output the correct outputBus (shown in hexadecimal) with the given inputs. This is aligned with our expectations.

FIFO_Control_testbench():

We chose to use FSM and pointer module to implement FIFO_Control, which allowed us to

Inputs:

1) clk
2) reset
3) read
4) write

Outputs:

1) wr_en
2) empty
3) full
4) [3:0] readAddr
5) [3:0] writeAddr

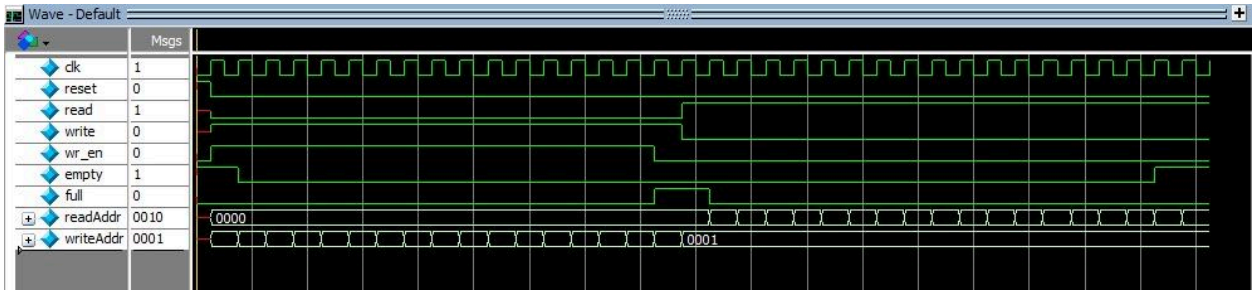

Figure 13: FIFO_Control testbench Modelsim

The image above shows a testbench for the FIFO_Control module. At the very beginning, when there is nothing in the FIFO, the empty signal is on and the full signal is off. As we write into FIFO, both empty and full are off. Moreover, wr_en is on when empty and write are true, and it is off when full is on even if the write signal is still on. After writing for 16 times, FIFO is full, which means the full signal is on. Then after we read for 16 times, the empty signal is on again. Waveform is consistent with our expectations for the functionality of FIFO FSM design.

Appendix

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 1
6
7         This module implements a memory-unit-like design which demonstrates
8      a 32x4 RAM chip
9         It uses switches on the DE1_SoC as input data for the RAM-like design
10        It outputs the user-input and stored memory values on the DE1_SoC HEXs
11
12     inputs:
13        All SW               switches on DE1_SoC
14        All KEY              keys on DE1_SoC
15     outputs:
16        [6:0] HEX:           7'b HEX display on DE1_SoC
17
18  */
19  module DE1_SoC(HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
20                 KEY, SW);
21     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
22     input logic [9:0] SW;
23     input logic [3:0] KEY;
24
25     logic [3:0] dout;
26     // Turn off Unused Displays
27     assign HEX1 = 7'b1111111;
28     assign HEX3 = 7'b1111111;
29
30     /* Reads din, and writes to the corresponding address
31        into internal memory, reads/returns the data stored
32        on given address when write is disabled */
33     M10K_32_4 mem (.clk(KEY[0]),.addr(SW[8:4]),.din(SW[3:0]),
34                 .write(SW[9]),.dout(dout));
35
36     /* Displays the given address on HEX 4-5 */
37     display_31digit disp_addr (.clk(KEY[0]), .count(SW[8:4]), .HEX0(HEX5), .HEX1(HEX4));
38
39     /* Displays the user input on HEX 2*/
40     HexadecimalConverter disp_din  (.clk(KEY[0]), .binary_input(SW[3:0]), .hexadecimal_output
    (HEX2));
41
42     /*Displays the data at specified address*/
43     HexadecimalConverter disp_dout  (.clk(KEY[0]), .binary_input(dout), .hexadecimal_output(
    HEX0));
44
45
46  endmodule                //De1_SoC
47  //-----------------------------------------------------------------
48  //DE1_SoC_testbench tests the expected and unexpected behaviors,
49  // reset behaviors, and random input behaviors
50
51  module DE1_SoC_testbench();
52     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
53     logic [9:0] SW;
54     logic [3:0] KEY;
55
56     DE1_SoC dut(HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
57                 KEY, SW);
58
59     // clock setup
60     parameter clock_period = 100;
61     initial begin
62        KEY[0] <= 0;
63        forever #(clock_period / 2) KEY[0] <= ~KEY[0];
64     end
65
66     initial begin
67        /*Display Address 15*/
68        SW[8:4] = 15;     @(posedge KEY[0]);
69        /*Write "0" to Address 15*/
70        SW[3:0] = 4'b0000;
71        SW[9] <= 1;          @(posedge KEY[0]);
72        SW[9] <= 0;          @(posedge KEY[0]);
73        /*Display Address 21*/
74        SW[8:4] = 21;     @(posedge KEY[0]);
```

```
75          /*Write "9" to Address 21*/
76          SW[3:0] = 9;
77          SW[9] <= 1;           @(posedge KEY[0]);
78          SW[9] <= 0;           @(posedge KEY[0]);
79
80          $stop;
81        end   //initial
82    endmodule               //De1_SoC_testbench
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 1
6
7         This module is a memory-unit-like design which models after a
8      a 32x4 RAM chip
9
10     inputs:
11        clk                  clock
12        [4:0] addr           address label
13        [3:0] din            data in
14        write                write enable
15     outputs:
16        [3:0] dout:          data out
17
18  */
19  module M10K_32_4(clk,addr,din,write,dout);
20     input logic clk;
21     input logic [4:0] addr;
22     input logic [3:0] din;
23     input logic write;
24     output logic [3:0] dout;
25
26     logic [3:0] mem [31:0];
27
28     always_ff @(posedge clk) begin
29         if (write)
30             // Write data to the specified address on the rising edge of the clock
31             mem[addr] <= din;
32     end
33
34     // Unregistered data output
35     assign dout = mem[addr];
36
37  endmodule               //M10K_32_4
38  /*-------------------------------------------------------------------------------------------*/
39  // M10K_32_4_testbench tests the expected and unexpected behaviors,
40  // reset behaviors, and random input behaviors
41
42  module M10K_32_4_testbench();
43     logic clk;
44     logic [4:0] addr;
45     logic [3:0] din;
46     logic write;
47     logic [3:0] dout;
48
49     M10K_32_4 dut(clk,addr,din,write,dout);
50
51     // clock setup
52     parameter clock_period = 100;
53     initial begin
54        clk <= 0;
55        forever #(clock_period / 2) clk <= ~clk;
56     end
57
58     initial begin
59        /*Write data to address 5*/
60        addr <= 5;
61        din <= 4'b1010;
62        write <= 1;    @(posedge clk)
63        write <= 0; @(posedge clk)
64
65        /*Read data from address 5*/
66        addr = 5;   @(posedge clk)
67        $stop;
68
69     end   //initial
70  endmodule               //M10K_32_4_testbench
```

```
305                         count = 5'b00101;       @(posedge clk);
306                                                 @(posedge clk);
307                         count = 5'b00110;       @(posedge clk);
308                                                 @(posedge clk);
309                         count = 5'b11001;       @(posedge clk);
310                                                 @(posedge clk);
311             $stop; // end simulation
312         end
313     endmodule                   //display_31digit_testbench
```

```
/*
Justin Sim and Mina Gao
1/23/2024
EE 371 Hussein
Lab 2

      This module will create a Hexadecimal rendering of input data for
   numbers between 0 and 31
      It takes in any 5'b input and returns the corresponding number for the
   DE1_SoC 7'b HEX displays

   inputs:
      clk                  clock
      [4:0] count          user input (SW)
   outputs:
      [6:0] HEX:           7'b HEX display on DE1_SoC

*/
module display_31digit (clk, count, HEX0, HEX1);
   input logic clk;
   input logic [4:0] count;
   output logic [6:0] HEX0, HEX1;

   //for 0-9
   logic [6:0] h0, h1, h2, h3, h4, h5, h6, h7, h8, h9, hOff;
// hC, hL, hE, hA, hR, hF, hU;

   assign h0 = 7'b1000000; // 0
   assign h1 = 7'b1111001; // 1
   assign h2 = 7'b0100100; // 2
   assign h3 = 7'b0110000; // 3
   assign h4 = 7'b0011001; // 4
   assign h5 = 7'b0010010; // 5
   assign h6 = 7'b0000010; // 6
   assign h7 = 7'b1111000; // 7
   assign h8 = 7'b0000000; // 8
   assign h9 = 7'b0010000; // 9
   assign hOff = 7'b1111111; // no display

   //for letters (clear, full)
// assign hC = 7'b1000110; // C
// assign hL = 7'b1000111; // L
// assign hE = 7'b0000110; // E
// assign hA = 7'b0001000; // A
// assign hR = 7'b0101111; // r
// assign hF = 7'b0001110; // F
// assign hU = 7'b1000001; // U

   always_comb begin
      case(count)
         // 0
         5'b00000:
            begin
                  HEX1 = hOff;
                  HEX0 = h0;
            end

         // 1
         5'b00001:
            begin
                  HEX1 = hOff;
                  HEX0 = h1;
            end

         // 2
         5'b00010:
            begin
                  HEX1 = hOff;
                  HEX0 = h2;
            end

         // 3
         5'b00011:
            begin
                  HEX1 = hOff;
                  HEX0 = h3;
```

```
77                end
78
79            // 4
80            5'b00100:
81               begin
82                     HEX1 = hOff;
83                     HEX0 = h4;
84               end
85
86            // 5
87            5'b00101:
88               begin
89                     HEX1 = hOff;
90                     HEX0 = h5;
91               end
92
93            // 6
94            5'b00110:
95               begin
96                     HEX1 = hOff;
97                     HEX0 = h6;
98               end
99
100           // 7
101           5'b00111:
102              begin
103                    HEX1 = hOff;
104                    HEX0 = h7;
105              end
106
107           // 8
108           5'b01000:
109              begin
110                    HEX1 = hOff;
111                    HEX0 = h8;
112              end
113
114           // 9
115           5'b01001:
116              begin
117                    HEX1 = hOff;
118                    HEX0 = h9;
119              end
120
121           // 10
122           5'b01010:
123              begin
124                    HEX1 = h1;
125                    HEX0 = h0;
126              end
127
128           // 11
129           5'b01011:
130              begin
131                    HEX1 = h1;
132                    HEX0 = h1;
133              end
134
135           // 12
136           5'b01100:
137              begin
138                    HEX1 = h2;
139                    HEX0 = h1;
140              end
141
142           // 13
143           5'b01101:
144              begin
145                    HEX1 = h3;
146                    HEX0 = h1;
147              end
148
149           // 14
150           5'b01110:
151              begin
152                    HEX1 = h4;
```

```
153                         HEX0 = h1;
154                 end
155
156             // 15
157             5'b01111:
158                 begin
159                         HEX1 = h5;
160                         HEX0 = h1;
161                 end
162
163             // 16
164             5'b10000:
165                 begin
166                         HEX1 = h6;
167                         HEX0 = h1;
168                 end
169
170             // 17
171             5'b10001:
172                 begin
173                         HEX1 = h7;
174                         HEX0 = h1;
175                 end
176
177             // 18
178             5'b10010:
179                 begin
180                         HEX1 = h8;
181                         HEX0 = h1;
182                 end
183
184             // 19
185             5'b10011:
186                 begin
187                         HEX1 = h9;
188                         HEX0 = h1;
189                 end
190
191             // 20
192             5'b10100:
193                 begin
194                         HEX1 = h0;
195                         HEX0 = h2;
196                 end
197
198             // 21
199             5'b10101:
200                 begin
201                         HEX1 = h1;
202                         HEX0 = h2;
203                 end
204
205             // 22
206             5'b10110:
207                 begin
208                         HEX1 = h2;
209                         HEX0 = h2;
210                 end
211
212             // 23
213             5'b10111:
214                 begin
215                         HEX1 = h3;
216                         HEX0 = h2;
217                 end
218
219             // 24
220             5'b11000:
221                 begin
222                         HEX1 = h4;
223                         HEX0 = h2;
224                 end
225
226             // 25
227             5'b11001:
228                 begin
```

```
229                         HEX1 = h5;
230                         HEX0 = h2;
231                     end
232
233                 // 26
234                 5'b11010:
235                     begin
236                         HEX1 = h6;
237                         HEX0 = h2;
238                     end
239
240                 // 27
241                 5'b11011:
242                     begin
243                         HEX1 = h7;
244                         HEX0 = h2;
245                     end
246                 // 28
247                 5'b11100:
248                     begin
249                         HEX1 = h8;
250                         HEX0 = h2;
251                     end
252                 // 29
253                 5'b11101:
254                     begin
255                         HEX1 = h9;
256                         HEX0 = h2;
257                     end
258                 // 30
259                 5'b11110:
260                     begin
261                         HEX1 = h0;
262                         HEX0 = h3;
263                     end
264                 // 31
265                 5'b11111:
266                     begin
267                         HEX1 = h1;
268                         HEX0 = h3;
269                     end
270             endcase
271         end
272     endmodule                //display_31digit
273     //-------------------------------------------------------------
274     // display_31digit_testbench tests the expected and unexpected behaviors,
275     // reset behaviors, and random input behaviors
276
277     module display_31digit_testbench ();
278         logic clk;
279         logic [6:0] HEX0, HEX1;
280         logic [4:0] count;
281
282         display_31digit dut (.clk(clk), .count(count), .HEX0(HEX0), .HEX1(HEX1));
283
284         // clock setup
285         parameter clock_period = 100;
286
287         initial begin
288             clk <= 0;
289             forever #(clock_period / 2) clk <= ~clk;
290         end
291
292         initial begin
293                                                 @(posedge clk);
294                                                 @(posedge clk);
295                         count = 5'b00000;       @(posedge clk);
296                                                 @(posedge clk);
297                         count = 5'b00001;       @(posedge clk);
298                                                 @(posedge clk);
299                         count = 5'b00010;       @(posedge clk);
300                                                 @(posedge clk);
301                         count = 5'b00011;       @(posedge clk);
302                                                 @(posedge clk);
303                         count = 5'b00100;       @(posedge clk);
304                                                 @(posedge clk);
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2
6
7         This module converts 4'b binary input to 7'b Hexadecimal for the DE1_SoC
8
9      inputs:
10        clk                  clock
11        [3:0] binary input   user input (SW)
12     outputs:
13        [7:0] hexadecimal_output:            7'b HEX display on DE1_SoC
14
15  */
16  module HexadecimalConverter  (clk, binary_input, hexadecimal_output);
17     input logic [3:0] binary_input;
18     input logic clk;
19     output logic [6:0] hexadecimal_output;
20     logic [6:0] h0, h1, h2, h3, h4, h5, h6, h7, h8, h9,
21                 hA, hB, hC, hD, hE, hF, hOff;
22
23     assign h0 = 7'b1000000; // 0
24     assign h1 = 7'b1111001; // 1
25     assign h2 = 7'b0100100; // 2
26     assign h3 = 7'b0110000; // 3
27     assign h4 = 7'b0011001; // 4
28     assign h5 = 7'b0010010; // 5
29     assign h6 = 7'b0000010; // 6
30     assign h7 = 7'b1111000; // 7
31     assign h8 = 7'b0000000; // 8
32     assign h9 = 7'b0010000; // 9
33     assign hA = 7'b0001000; // A
34     assign hB = 7'b0000011; // B
35     assign hC = 7'b1000110; // C
36     assign hD = 7'b0100001; // D
37     assign hE = 7'b0000110; // E
38     assign hF = 7'b0001110; // F
39     assign hOff = 7'b1111111; // no display
40
41    always_comb begin
42      case (binary_input)
43        4'b0000: hexadecimal_output = h0;
44        4'b0001: hexadecimal_output = h1;
45        4'b0010: hexadecimal_output = h2;
46        4'b0011: hexadecimal_output = h3;
47        4'b0100: hexadecimal_output = h4;
48        4'b0101: hexadecimal_output = h5;
49        4'b0110: hexadecimal_output = h6;
50        4'b0111: hexadecimal_output = h7;
51        4'b1000: hexadecimal_output = h8;
52        4'b1001: hexadecimal_output = h9;
53        4'b1010: hexadecimal_output = hA;
54        4'b1011: hexadecimal_output = hB;
55        4'b1100: hexadecimal_output = hC;
56        4'b1101: hexadecimal_output = hD;
57        4'b1110: hexadecimal_output = hE;
58        4'b1111: hexadecimal_output = hF;
59        4'bXXXX: hexadecimal_output = hOff; // Handle other cases as needed
60      endcase
61    end
62
63  endmodule                // HexadecimalConverter
64  //-------------------------------------------------------------
65  //HexadecimalConverter_testbench tests the expected and unexpected behaviors,
66  // reset behaviors, and random input behaviors
67
68  module HexadecimalConverter_testbench ;
69
70    logic [3:0] binary_input;
71    logic [7:0] hexadecimal_output;
72    logic clock;
73
74    // Instantiate HexadecimalConverter module
75    HexadecimalConverter  dut (
76      .binary_input(binary_input),
```

```
77        .hexadecimal_output(hexadecimal_output)
78      );
79
80      // clock setup
81       parameter clock_period = 100;
82       initial begin
83          clock <= 0;
84          forever #(clock_period / 2) clock <= ~clock;
85       end
86
87      // Test stimulus
88      initial begin
89        binary_input = 4'b0000;       @(posedge clock);
90
91        // Apply test vectors
92        binary_input = 4'b1010;    @(posedge clock);// Binary input: 1010 (Decimal: 10)
93        binary_input = 4'b1101;    @(posedge clock);// Binary input: 1101 (Decimal: 13)
94        binary_input = 4'b0111;    @(posedge clock);// Binary input: 0111 (Decimal: 7)
95
96        // Add more test vectors as needed
97
98       $stop; // Stop simulation after a certain duration
99      end
100
101   endmodule               // HexadecimalConverter_testbench
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 2
6
7      This module will keep track of how many cars have entered
8      or exited the garage.
9      It uses two sensors, a and b, to detect vehicle
10     movement in and out of the garage, implements an fsm and counter.
11     The output will be on the HEX displaying the number of cars
12     currently occupying the garage, and will say FULL when
13     the maximum capacity (editable parameter) has been reached
14
15     inputs:
16        CLOCK_50
17        [35:0] V_GPIO:       Virtual GPIO for DE1_SoC
18     in/outputs:
19        [6:0] HEX:           7'b HEX display on DE1_SoC
20
21  */
22  module DE1_SoC(CLOCK_50, KEY, SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);
23     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
24     input logic [9:0] SW;
25     input logic [3:0] KEY;
26     input logic CLOCK_50;
27
28     logic [4:0]  rdaddress;
29     logic [3:0] dout;
30     logic reset;
31     assign reset = ~KEY[0];
32
33     logic [31:0] div_clk;
34     parameter whichClock = 25; // 0.75 Hz clock
35     clock_divider cdiv (.clock(CLOCK_50),
36     .reset(reset),
37     .divided_clocks(div_clk));
38     // Clock selection; allows for easy switching between simulation and boardclocks
39     logic clkSelect;
40
41     // Uncomment ONE of the following two lines depending on intention
42     assign clkSelect = CLOCK_50; // for simulation
43  // assign clkSelect = div_clk[whichClock]; // for board
44
45     // Instantiate the memory with MIF
46     ram32x4 ram (
47     .clock(CLOCK_50),
48     .data(SW[3:0]),
49     .rdaddress(rdaddress),
50     .wraddress(SW[8:4]),
51     .wren(~KEY[3]),
52     .q(dout));
53
54     /* Increment rdaddress by 1 every second, using clkSelect (1 sec)*/
55     counter ct (.clk(clkSelect), .reset(reset), .rdaddress(rdaddress));
56
57     /* Displays the write address on HEX 4-5 */
58     display_31digit disp_wr_addr (.clk(CLOCK_50), .count(SW[8:4]), .HEX0(HEX5), .HEX1(HEX4));
59
60     /* Displays the read address on HEX 2-3 */
61     display_31digit disp_rd_addr (.clk(CLOCK_50), .count(rdaddress), .HEX0(HEX3), .HEX1(HEX2
    ));
62
63     /* Displays the user input on HEX 1*/
64     HexadecimalConverter disp_din  (.clk(CLOCK_50), .binary_input(SW[3:0]), .
    hexadecimal_output(HEX1));
65
66     /*Displays the data at specified address*/
67     HexadecimalConverter disp_dout  (.clk(CLOCK_50), .binary_input(dout), .hexadecimal_output
    (HEX0));
68
69  endmodule               // DE1_SoC
70  //-------------------------------------------------------------------------------------------
    -------------
71  //DE1_SoC_testbench tests the expected and unexpected behaviors, reset behaviors
72  // specifically, it tests a full cycle of scrolling through addresses
```

```
73   // and shows a case of the RAM's write function
74
75   `timescale 1 ps / 1 ps
76   module DE1_SoC_testbench();
77      logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
78      logic [9:0] SW;
79      logic [3:0] KEY;
80      logic CLOCK_50;
81
82      DE1_SoC dut(.CLOCK_50(CLOCK_50), .HEX0(HEX0), .HEX1(HEX1),
83      .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5), .KEY(KEY), .SW(SW));
84
85      // clock setup
86      parameter clock_period = 100;
87      initial begin
88         CLOCK_50 <= 0;
89         forever #(clock_period / 2) CLOCK_50 <= ~CLOCK_50;
90      end
91
92         integer i;
93         integer n;
94
95      initial begin
96         // count up
97         KEY[0] = 0; @(posedge CLOCK_50);
98         KEY[0] = 1; @(posedge CLOCK_50);
99         for (n=1; n<=31; n++) begin
100        @(posedge CLOCK_50);
101        end
102
103        // write "0" to address 4
104
105        SW[8:4] = 5'b00100;
106        SW[3:0] = 4'b0000;   @(posedge CLOCK_50); //write enable
107        KEY[3] = 0; @(posedge CLOCK_50); @(posedge CLOCK_50);
108        KEY[3] = 1; @(posedge CLOCK_50);
109        @(posedge CLOCK_50);
110        @(posedge CLOCK_50);
111        @(posedge CLOCK_50);
112        @(posedge CLOCK_50);
113        $stop;
114     end   //initial
115  endmodule                     // DE1_SoC_testbench
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 2
6
7      This module divides the input clock such that it can slow down the cycle:
8      divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz,[25] = 0.75Hz, ...
9
10     inputs:
11        clock
12     in/outputs:
13        [31:0] divided_clocks            array of varying clock speeds
14
15  */
16  module clock_divider (clock, reset, divided_clocks);
17     input logic reset, clock;
18     output logic [31:0] divided_clocks = 0;
19     always_ff @(posedge clock) begin
20        divided_clocks <= divided_clocks + 1;
21     end
22  endmodule               // clock_divider
23  //-------------------------------------------------------------------------------------------
    -------------
24  // clock_divider_testbench tests the expected behaviors of the clock_divider module
25
26  module clock_divider_testbench();
27  logic reset, clock;
28  logic [31:0] divided_clocks;
29
30  clock_divider dut (.clock, .reset, .divided_clocks);
31
32  // clock setup
33     parameter clock_period = 100;
34     initial begin
35        clock <= 0;
36        forever #(clock_period / 2) clock <= ~clock;
37     end
38     integer n;
39     initial begin
40        for (n=1; n<=31; n++) begin
41        @(posedge clock);
42        end
43        reset = 1;
44        $stop;
45     end
46
47  endmodule               // clock_divider_testbench
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 2
6
7      This module will provide a count up from 0 to 31
8      starts the count back at 0 when reset
9      will automatically cycle back to 0 after reaching 31
10
11     inputs:
12        clk                  clock
13        reset
14     in/outputs:
15        [4:0] rdaddress      read address
16
17  */
18  module counter (clk, reset, rdaddress);
19     input logic clk, reset;
20     output logic [4:0] rdaddress;
21
22     always_ff @(posedge clk) begin
23        if (reset) rdaddress <= 5'b00000;
24
25        else begin
26           if (rdaddress == 5'b11111) // edge case
27              rdaddress <= 5'b00000;
28           else
29              rdaddress <= rdaddress + 5'd00001;
30        end
31     end
32  endmodule               // counter
33  //-------------------------------------------------------------------------------------------
    -------------
34  // counter_testbench tests the expected and unexpected behaviors, reset behaviors
35  // specifically, it tests a full cycle from 0 - 31
36
37  module counter_testbench();
38     logic clk, reset;
39     logic [4:0] rdaddress;
40
41     counter dut (.clk, .reset, .rdaddress);
42
43     // clock setup
44     parameter clock_period = 100;
45
46     initial begin
47        clk <= 0;
48        forever #(clock_period / 2) clk <= ~clk;
49     end
50
51     integer i;
52
53     initial begin
54        reset <= 0;             @(posedge clk);
55        reset <= 1;             @(posedge clk);
56        reset <= 0; repeat(34)  @(posedge clk);
57        reset <= 1; @(posedge clk)
58        reset <= 0; repeat(5)   @(posedge clk);
59        $stop;
60     end
61
62  endmodule               // counter_testbench
```

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0	1	2	3	4	5	6	7	.....
8	0	2	4	6	8	10	12	14	.....
16	0	1	3	5	7	9	11	13	.....
24	7	6	5	4	3	2	1	0	.....

```
153   // Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
154   // Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "0"
155   // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
156   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
157   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
158   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
159   // Retrieval info: PRIVATE: REGdata NUMERIC "1"
160   // Retrieval info: PRIVATE: REGq NUMERIC "1"
161   // Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
162   // Retrieval info: PRIVATE: REGrren NUMERIC "1"
163   // Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
164   // Retrieval info: PRIVATE: REGwren NUMERIC "1"
165   // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
166   // Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
167   // Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
168   // Retrieval info: PRIVATE: VarWidth NUMERIC "0"
169   // Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "4"
170   // Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "4"
171   // Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "4"
172   // Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "4"
173   // Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
174   // Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "0"
175   // Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
176   // Retrieval info: PRIVATE: enable NUMERIC "0"
177   // Retrieval info: PRIVATE: rden NUMERIC "0"
178   // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
179   // Retrieval info: CONSTANT: ADDRESS_ACLR_B STRING "NONE"
180   // Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
181   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
182   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
183   // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
184   // Retrieval info: CONSTANT: INIT_FILE STRING "ram32x4.mif"
185   // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
186   // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
187   // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32"
188   // Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "32"
189   // Retrieval info: CONSTANT: OPERATION_MODE STRING "DUAL_PORT"
190   // Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
191   // Retrieval info: CONSTANT: OUTDATA_REG_B STRING "UNREGISTERED"
192   // Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
193   // Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
194   // Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING "DONT_CARE"
195   // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "5"
196   // Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "5"
197   // Retrieval info: CONSTANT: WIDTH_A NUMERIC "4"
198   // Retrieval info: CONSTANT: WIDTH_B NUMERIC "4"
199   // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
200   // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
201   // Retrieval info: USED_PORT: data 0 0 4 0 INPUT NODEFVAL "data[3..0]"
202   // Retrieval info: USED_PORT: q 0 0 4 0 OUTPUT NODEFVAL "q[3..0]"
203   // Retrieval info: USED_PORT: rdaddress 0 0 5 0 INPUT NODEFVAL "rdaddress[4..0]"
204   // Retrieval info: USED_PORT: wraddress 0 0 5 0 INPUT NODEFVAL "wraddress[4..0]"
205   // Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
206   // Retrieval info: CONNECT: @address_a 0 0 5 0 wraddress 0 0 5 0
207   // Retrieval info: CONNECT: @address_b 0 0 5 0 rdaddress 0 0 5 0
208   // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
209   // Retrieval info: CONNECT: @data_a 0 0 4 0 data 0 0 4 0
210   // Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
211   // Retrieval info: CONNECT: q 0 0 4 0 @q_b 0 0 4 0
212   // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.v TRUE
213   // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.inc FALSE
214   // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.cmp FALSE
215   // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.bsf FALSE
216   // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_inst.v FALSE
217   // Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_bb.v TRUE
218   // Retrieval info: LIB_FILE: altera_mf
219
```

```
1   // megafunction wizard: %RAM: 2-PORT%
2   // GENERATION: STANDARD
3   // VERSION: WM1.0
4   // MODULE: altsyncram
5
6   // ============================================================
7   // File Name: ram32x4.v
8   // Megafunction Name(s):
9   //          altsyncram
10  //
11  // Simulation Library Files(s):
12  //          altera_mf
13  // ============================================================
14  // ************************************************************
15  // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
16  //
17  // 17.0.0 Build 595 04/25/2017 SJ Lite Edition
18  // ************************************************************
19
20
21  //Copyright (C) 2017  Intel Corporation. All rights reserved.
22  //Your use of Intel Corporation's design tools, logic functions
23  //and other software and tools, and its AMPP partner logic
24  //functions, and any output files from any of the foregoing
25  //(including device programming or simulation files), and any
26  //associated documentation or information are expressly subject
27  //to the terms and conditions of the Intel Program License
28  //Subscription Agreement, the Intel Quartus Prime License Agreement,
29  //the Intel MegaCore Function License Agreement, or other
30  //applicable license agreement, including, without limitation,
31  //that your use is for the sole purpose of programming logic
32  //devices manufactured by Intel and sold by Intel or its
33  //authorized distributors.  Please refer to the applicable
34  //agreement for further details.
35
36
37  // synopsys translate_off
38  `timescale 1 ps / 1 ps
39  // synopsys translate_on
40  module ram32x4 (
41     clock,
42     data,
43     rdaddress,
44     wraddress,
45     wren,
46     q);
47
48     input    clock;
49     input [3:0]  data;
50     input [4:0]  rdaddress;
51     input [4:0]  wraddress;
52     input    wren;
53     output   [3:0]  q;
54  `ifndef ALTERA_RESERVED_QIS
55  // synopsys translate_off
56  `endif
57     tri1     clock;
58     tri0     wren;
59  `ifndef ALTERA_RESERVED_QIS
60  // synopsys translate_on
61  `endif
62
63     wire [3:0] sub_wire0;
64     wire [3:0] q = sub_wire0[3:0];
65
66     altsyncram  altsyncram_component (
67               .address_a (wraddress),
68               .address_b (rdaddress),
69               .clock0 (clock),
70               .data_a (data),
71               .wren_a (wren),
72               .q_b (sub_wire0),
73               .aclr0 (1'b0),
74               .aclr1 (1'b0),
75               .addressstall_a (1'b0),
76               .addressstall_b (1'b0),
```

```
77                .byteena_a (1'b1),
78                .byteena_b (1'b1),
79                .clock1 (1'b1),
80                .clocken0 (1'b1),
81                .clocken1 (1'b1),
82                .clocken2 (1'b1),
83                .clocken3 (1'b1),
84                .data_b ({4{1'b1}}),
85                .eccstatus (),
86                .q_a (),
87                .rden_a (1'b1),
88                .rden_b (1'b1),
89                .wren_b (1'b0));
90        defparam
91           altsyncram_component.address_aclr_b = "NONE",
92           altsyncram_component.address_reg_b = "CLOCK0",
93           altsyncram_component.clock_enable_input_a = "BYPASS",
94           altsyncram_component.clock_enable_input_b = "BYPASS",
95           altsyncram_component.clock_enable_output_b = "BYPASS",
96           altsyncram_component.init_file = "ram32x4.mif",
97           altsyncram_component.intended_device_family = "Cyclone V",
98           altsyncram_component.lpm_type = "altsyncram",
99           altsyncram_component.numwords_a = 32,
100          altsyncram_component.numwords_b = 32,
101          altsyncram_component.operation_mode = "DUAL_PORT",
102          altsyncram_component.outdata_aclr_b = "NONE",
103          altsyncram_component.outdata_reg_b = "UNREGISTERED",
104          altsyncram_component.power_up_uninitialized = "FALSE",
105          altsyncram_component.ram_block_type = "M10K",
106          altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
107          altsyncram_component.widthad_a = 5,
108          altsyncram_component.widthad_b = 5,
109          altsyncram_component.width_a = 4,
110          altsyncram_component.width_b = 4,
111          altsyncram_component.width_byteena_a = 1;
112
113
114   endmodule
115
116   // ============================================================
117   // CNX file retrieval info
118   // ============================================================
119   // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
120   // Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
121   // Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
122   // Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
123   // Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
124   // Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
125   // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
126   // Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
127   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
128   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
129   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
130   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
131   // Retrieval info: PRIVATE: CLRdata NUMERIC "0"
132   // Retrieval info: PRIVATE: CLRq NUMERIC "0"
133   // Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
134   // Retrieval info: PRIVATE: CLRrren NUMERIC "0"
135   // Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
136   // Retrieval info: PRIVATE: CLRwren NUMERIC "0"
137   // Retrieval info: PRIVATE: Clock NUMERIC "0"
138   // Retrieval info: PRIVATE: Clock_A NUMERIC "0"
139   // Retrieval info: PRIVATE: Clock_B NUMERIC "0"
140   // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
141   // Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
142   // Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
143   // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
144   // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
145   // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
146   // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
147   // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
148   // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
149   // Retrieval info: PRIVATE: MEMSIZE NUMERIC "128"
150   // Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
151   // Retrieval info: PRIVATE: MIFfilename STRING "ram32x4.mif"
152   // Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
```

```
1   /*
2   Mina Gao and Justin Sim
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 3
6
7      This module implements a synchronous FIFO mechanism
8      Displays the value of user input and the current data ouput in hexadecimal om HEX display
9      Uses LEDR8 and LEDR9 to represent the state of the FIFO
10
11     inputs:
12        CLOCK_50
13        All SW                switches on DE1_SoC
14        All KEY               keys on DE1_SoC
15     outputs:
16        [6:0] HEX             7'b HEX display on DE1_SoC
17        [9:0] LEDR            LED on DE1_SoC
18
19  */
20  module DE1_SoC(CLOCK_50, KEY, SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, LEDR);
21     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
22     output logic [9:0] LEDR;
23     input logic [9:0] SW;
24     input logic [3:0] KEY;
25     input logic CLOCK_50;
26
27     logic [7:0] inputBus, outputBus;
28
29     /* Turn off HEX2 and HEX3 */
30     assign HEX2 = 7'b1111111;
31     assign HEX3 = 7'b1111111;
32
33     /* Displays the data input on HEX 5 */
34     HexadecimalConverter disp_in_h5 (.clk(CLOCK_50), .binary_input(SW[7:4]), .
   hexadecimal_output(HEX5));
35
36     /* Displays the data input on HEX 4 */
37     HexadecimalConverter disp_in_h4 (.clk(CLOCK_50), .binary_input(SW[3:0]), .
   hexadecimal_output(HEX4));
38
39     /* Displays the current data output on HEX 1*/
40     HexadecimalConverter disp_out_h1 (.clk(CLOCK_50), .binary_input(outputBus[7:4]), .
   hexadecimal_output(HEX1));
41
42     /* Displays the current data output on HEX 0*/
43     HexadecimalConverter disp_out_h0 (.clk(CLOCK_50), .binary_input(outputBus[3:0]), .
   hexadecimal_output(HEX0));
44
45     /* LEDR9: full, LEDR8: empty, 50MHz clock*/
46     FIFO queue (.clk(CLOCK_50), .reset(~KEY[0]), .read(~KEY[3]), .write(~KEY[2]), .inputBus(
   SW[7:0]), .empty(LEDR[8]), .full(LEDR[9]), .outputBus);
47  endmodule                //DE1_SoC
48  //-------------------------------------------------------------------------------------------
   -------------
49  //DE1_SoC_testbench tests the expected and unexpected behaviors, reset behaviors
50
51  `timescale 1 ps / 1 ps
52  module DE1_SoC_testbench();
53     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
54     logic [9:0] LEDR;
55     logic [9:0] SW;
56     logic [3:0] KEY;
57     logic CLOCK_50;
58
59     logic reset, write, read;
60     assign KEY[0] = ~reset;
61     assign KEY[2] = ~write;
62     assign KEY[3] = ~read;
63
64     logic [7:0] inputBus;
65     assign SW[7:0] = inputBus;
66
67     DE1_SoC dut(.CLOCK_50(CLOCK_50), .HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .SW, .
   LEDR);
68
69     // clock setup
```

```
70       parameter clock_period = 100;
71       initial begin
72          CLOCK_50 <= 0;
73          forever #(clock_period / 2) CLOCK_50 <= ~CLOCK_50;
74       end
75
76       integer i;
77
78       initial begin
79          reset <= 1;                                          @(posedge CLOCK_50);
80                                                               @(posedge CLOCK_50);
81          reset <= 0;                                          @(posedge CLOCK_50); //keep writing for
     18 times, until memory is full
82          for (i=0; i<=17; i++) begin
83                read <= 0; write <= 1;
84                inputBus <= 8'h00 + i;                         @(posedge CLOCK_50);
85          end
86          read <= 1; write <= 0;              repeat(18) @(posedge CLOCK_50); //then read for 18
     times
87          $stop; // end simulation
88       end   //initial
89    endmodule               //DE1_SoC
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 3
6
7         This module implements a memory-unit-like design which demonstrates
8      a 16x8 RAM chip
9         It uses switches on the DE1_SoC as input data for the RAM-like design
10        It outputs the user-input and stored memory values on the DE1_SoC HEXs
11
12     inputs:
13        clk, reset
14        read, write
15        [width-1:0] inputBus
16     outputs:
17        empty, full
18        [width-1:0] outputBus
19
20  */
21  module FIFO #(
22                parameter depth = 4,
23                parameter width = 8
24                )(
25                  input logic clk, reset,
26                  input logic read, write,
27                  input logic [width-1:0] inputBus,
28                 output logic empty, full,
29                 output logic [width-1:0] outputBus
30                 );
31
32     /*    Define_Variables_Here      */
33     logic [3:0] r_address, w_address;
34     logic w_en;
35
36     /*       Instantiate_Your_Dual-Port_RAM_Here        */
37     ram16x8 ram (
38     .clock(clk),
39     .data(inputBus),
40     .rdaddress(r_address),
41     .wraddress(w_address),
42     .wren(w_en),
43     .q(outputBus));
44
45
46     /*       FIFO-Control_Module        */
47     FIFO_Control #(depth) FC (.clk, .reset,
48                               .read,
49                               .write,
50                               .wr_en(w_en),
51                               .empty,
52                               .full,
53                               .readAddr(r_address),
54                               .writeAddr(w_address)
55                              );
56
57  endmodule               //FIFO
58  //-------------------------------------------------------------------------------------------
    --------------
59  //FIFO_testbench tests the expected and unexpected behaviors, reset behaviors
60
61  `timescale 1 ps / 1 ps
62  module FIFO_testbench();
63
64     parameter depth = 4, width = 8;
65
66     logic clk, reset;
67     logic read, write;
68     logic [width-1:0] inputBus;
69     logic resetState;
70     logic empty, full;
71     logic [width-1:0] outputBus;
72
73     FIFO #(depth, width) dut (.*);
74
75     parameter CLK_Period = 100;
```

```
 76
 77      initial begin
 78         clk <= 1'b0;
 79         forever #(CLK_Period/2) clk <= ~clk;
 80      end
 81
 82      integer i;
 83
 84      initial begin
 85         reset <= 1;                                                    @(posedge clk);
 86         reset <= 0; read <= 0; write <= 1; inputBus <= 8'h00;    @(posedge clk);
 87                     read <= 0; write <= 1; inputBus <= 8'h01;    @(posedge clk); //keep
    writing for 18 times, until memory is full
 88                     read <= 0; write <= 1; inputBus <= 8'h02;    @(posedge clk);
 89                     read <= 0; write <= 1; inputBus <= 8'h03;    @(posedge clk);
 90                     read <= 0; write <= 1; inputBus <= 8'h04;    @(posedge clk);
 91                     read <= 0; write <= 1; inputBus <= 8'h05;    @(posedge clk);
 92                     read <= 0; write <= 1; inputBus <= 8'h06;    @(posedge clk);
 93                     read <= 0; write <= 1; inputBus <= 8'h07;    @(posedge clk);
 94                     read <= 0; write <= 1; inputBus <= 8'h08;    @(posedge clk);
 95                     read <= 0; write <= 1; inputBus <= 8'h09;    @(posedge clk);
 96                     read <= 0; write <= 1; inputBus <= 8'h0a;    @(posedge clk);
 97                     read <= 0; write <= 1; inputBus <= 8'h0b;    @(posedge clk);
 98                     read <= 0; write <= 1; inputBus <= 8'h0c;    @(posedge clk);
 99                     read <= 0; write <= 1; inputBus <= 8'h0d;    @(posedge clk);
100                     read <= 0; write <= 1; inputBus <= 8'h0e;    @(posedge clk);
101                     read <= 0; write <= 1; inputBus <= 8'h0f;    @(posedge clk);
102                     read <= 0; write <= 1; inputBus <= 8'h10;    @(posedge clk);
103                     read <= 0; write <= 1; inputBus <= 8'h11;    @(posedge clk);
104                     read <= 1; write <= 0;             repeat(18) @(posedge clk); //then read
    for 18 times
105     //    for (i=0; i<=17; i++) begin
106     //          read <= 0; write <= 1;
107     //          inputBus <= 8'h00 + 1'b1;                  @(posedge clk);
108     //    end
109           $stop; // end simulation
110      end
111     endmodule               //FIFO_testbench
112
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 3
6
7         This module implements a synchronous FIFO design using FSM
8         There are 3 states: S0 (empty), S1 (neither), S2(full)
9         It outputs wr_en, states of the FSM, read address, and write address
10
11     inputs:
12        clk, reset
13        read, write
14     outputs:
15        wr_en                                write enable
16        empty, full                          cases
17        [depth-1:0] readAddr, writeAddr      read address, write address
18
19  */
20  module FIFO_Control #(
21                        parameter depth = 4
22                        )(
23                          input logic clk, reset,
24                          input logic read, write,
25                         output logic wr_en,
26                         output logic empty, full,
27                         output logic [depth-1:0] readAddr, writeAddr
28                         );
29
30     /*    Define_Variables_Here      */
31     pointer rAdd (.clk(clk), .reset(reset), .RorW(read), .ptr_RorW(readAddr));
32     pointer wAdd (.clk(clk), .reset(reset), .RorW(write), .ptr_RorW(writeAddr));
33
34     // S0: empty
35     // S1: neither
36     // S2: full
37     enum {S0, S1, S2} ps, ns;
38
39     /*    Combinational_Logic_Here    */
40     always_comb begin
41           case(ps)
42              S0:   if (write && !read)                                                  ns = S1;
43                    else                                                                ns = ps;
44              S1:   if ((write && !read) && (readAddr == writeAddr - 1'b1))             ns = S2;
45                    else if ((!write && read) && (writeAddr == readAddr - 1'b1))        ns = S0;
46                    else                                                                ns = ps;
47              S2:   if (!write && read)                                                  ns = S1;
48                    else                                                                ns = ps;
49           endcase
50     end
51
52     always_comb begin
53           case(ps)
54              S0:   begin
55                    empty = 1'b1;
56                    full = 1'b0;
57                    end
58              S1:   begin
59                    empty = 1'b0;
60                    full = 1'b0;
61                    end
62              S2:   begin
63                    empty = 1'b0;
64                    full = 1'b1;
65                    end
66           endcase
67           if (write && ~full)  wr_en = 1'b1;
68           else                 wr_en = 1'b0;
69     end
70
71     /*    Sequential_Logic_Here      */
72     always_ff @(posedge clk) begin
73        if(reset) begin
74  //         readAddr <= '0;
75  //        writeAddr <= '0;
76                  ps <= S0;
```

```
77              // empty <= 1'b1;
78               // full <= 1'b0;
79          end else begin
80              ps <= ns;
81          end
82       end
83    endmodule                //FIFO_Control
84    //------------------------------------------------------------------------------------------
      --------------
85    //FIFO_Control_testbench tests the expected and unexpected behaviors, reset behaviors
86
87    module FIFO_Control_testbench ();
88       logic clk, reset, read, write;
89       logic wr_en, empty, full;
90       logic [3:0] readAddr, writeAddr;
91
92       FIFO_Control FC (.clk, .reset, .read, .write, .wr_en, .empty, .full, .readAddr, .
      writeAddr);
93
94       // clock setup
95       parameter clock_period = 100;
96
97       initial begin
98          clk <= 0;
99          forever #(clock_period / 2) clk <= ~clk;
100      end
101
102      initial begin
103         reset <= 1;                                          @(posedge clk);
104         reset <= 0; read <= 0; write <= 1; repeat(18) @(posedge clk); //keep writing for 18
      times
105                     read <= 1; write <= 0; repeat(18) @(posedge clk); //keep reading for 18
      times
106                                                          @(posedge clk);
107         $stop; // end simulation
108      end
109   endmodule                //FIFO_Control_testbench
110
```

```
1   /*
2   Justin Sim and Mina Gao
3   1/23/2024
4   EE 371 Hussein
5   Lab 2, Task 3
6
7         This module updates the read or write address base on the given signal (read or write)
8         It outputs the current read or write address to the FIFO_Control module
9
10     inputs:
11        clk, reset
12        RorW                  read or write
13     outputs:
14        [3:0] ptr_RorW        pointer read or write output
15
16  */
17  module pointer (clk, reset, RorW, ptr_RorW);
18     input logic clk, reset;
19     input logic RorW;
20     output logic [3:0] ptr_RorW;
21
22     always_ff @(posedge clk) begin
23        if (reset)                                ptr_RorW <= 4'b0000;
24        else if (RorW & ptr_RorW == 4'b1111)      ptr_RorW <= 4'b0000;
25        else if (RorW)                            ptr_RorW <= ptr_RorW + 4'd0001;
26        else                                      ptr_RorW <= ptr_RorW;
27     end
28  endmodule                //pointer
29  //-------------------------------------------------------------------------------------------
    -------------
30  //pointer_testbench tests the expected and unexpected behaviors, reset behaviors
31
32  module pointer_testbeench ();
33     logic clk, reset;
34     logic RorW;
35     logic [3:0] ptr_RorW;
36
37     pointer dut (.clk, .reset, .RorW, .ptr_RorW);
38
39     // clock setup
40     parameter clock_period = 100;
41
42     initial begin
43        clk <= 0;
44        forever #(clock_period / 2) clk <= ~clk;
45     end
46
47     initial begin
48        reset <= 1;                          @(posedge clk);
49                                             @(posedge clk);
50        reset <= 0; RorW <= 1; repeat(18) @(posedge clk); //keep increasing until passing the
    upper bond
51                                             @(posedge clk);
52        $stop; // end simulation
53     end
54  endmodule                //pointer_testbench
55
```

```
1   // megafunction wizard: %RAM: 2-PORT%
2   // GENERATION: STANDARD
3   // VERSION: WM1.0
4   // MODULE: altsyncram
5
6   // ============================================================
7   // File Name: ram16x8.v
8   // Megafunction Name(s):
9   //           altsyncram
10  //
11  // Simulation Library Files(s):
12  //           altera_mf
13  // ============================================================
14  // ************************************************************
15  // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
16  //
17  // 17.0.0 Build 595 04/25/2017 SJ Lite Edition
18  // ************************************************************
19
20
21  //Copyright (C) 2017  Intel Corporation. All rights reserved.
22  //Your use of Intel Corporation's design tools, logic functions
23  //and other software and tools, and its AMPP partner logic
24  //functions, and any output files from any of the foregoing
25  //(including device programming or simulation files), and any
26  //associated documentation or information are expressly subject
27  //to the terms and conditions of the Intel Program License
28  //Subscription Agreement, the Intel Quartus Prime License Agreement,
29  //the Intel MegaCore Function License Agreement, or other
30  //applicable license agreement, including, without limitation,
31  //that your use is for the sole purpose of programming logic
32  //devices manufactured by Intel and sold by Intel or its
33  //authorized distributors.  Please refer to the applicable
34  //agreement for further details.
35
36
37  // synopsys translate_off
38  `timescale 1 ps / 1 ps
39  // synopsys translate_on
40  module ram16x8 (
41     clock,
42     data,
43     rdaddress,
44     wraddress,
45     wren,
46     q);
47
48     input    clock;
49     input [7:0]  data;
50     input [3:0]  rdaddress;
51     input [3:0]  wraddress;
52     input    wren;
53     output   [7:0]  q;
54  `ifndef ALTERA_RESERVED_QIS
55  // synopsys translate_off
56  `endif
57     tri1     clock;
58     tri0     wren;
59  `ifndef ALTERA_RESERVED_QIS
60  // synopsys translate_on
61  `endif
62
63     wire [7:0] sub_wire0;
64     wire [7:0] q = sub_wire0[7:0];
65
66     altsyncram  altsyncram_component (
67               .address_a (wraddress),
68               .address_b (rdaddress),
69               .clock0 (clock),
70               .data_a (data),
71               .wren_a (wren),
72               .q_b (sub_wire0),
73               .aclr0 (1'b0),
74               .aclr1 (1'b0),
75               .addressstall_a (1'b0),
76               .addressstall_b (1'b0),
```

```
77                .byteena_a (1'b1),
78                .byteena_b (1'b1),
79                .clock1 (1'b1),
80                .clocken0 (1'b1),
81                .clocken1 (1'b1),
82                .clocken2 (1'b1),
83                .clocken3 (1'b1),
84                .data_b ({8{1'b1}}),
85                .eccstatus (),
86                .q_a (),
87                .rden_a (1'b1),
88                .rden_b (1'b1),
89                .wren_b (1'b0));
90        defparam
91           altsyncram_component.address_aclr_b = "NONE",
92           altsyncram_component.address_reg_b = "CLOCK0",
93           altsyncram_component.clock_enable_input_a = "BYPASS",
94           altsyncram_component.clock_enable_input_b = "BYPASS",
95           altsyncram_component.clock_enable_output_b = "BYPASS",
96           altsyncram_component.init_file = "ram16x8.mif",
97           altsyncram_component.intended_device_family = "Cyclone V",
98           altsyncram_component.lpm_type = "altsyncram",
99           altsyncram_component.numwords_a = 16,
100          altsyncram_component.numwords_b = 16,
101          altsyncram_component.operation_mode = "DUAL_PORT",
102          altsyncram_component.outdata_aclr_b = "NONE",
103          altsyncram_component.outdata_reg_b = "CLOCK0",
104          altsyncram_component.power_up_uninitialized = "FALSE",
105          altsyncram_component.ram_block_type = "M10K",
106          altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
107          altsyncram_component.widthad_a = 4,
108          altsyncram_component.widthad_b = 4,
109          altsyncram_component.width_a = 8,
110          altsyncram_component.width_b = 8,
111          altsyncram_component.width_byteena_a = 1;
112
113
114   endmodule
115
116   // ============================================================
117   // CNX file retrieval info
118   // ============================================================
119   // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
120   // Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
121   // Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
122   // Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
123   // Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
124   // Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
125   // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
126   // Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
127   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
128   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
129   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
130   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
131   // Retrieval info: PRIVATE: CLRdata NUMERIC "0"
132   // Retrieval info: PRIVATE: CLRq NUMERIC "0"
133   // Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
134   // Retrieval info: PRIVATE: CLRrren NUMERIC "0"
135   // Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
136   // Retrieval info: PRIVATE: CLRwren NUMERIC "0"
137   // Retrieval info: PRIVATE: Clock NUMERIC "0"
138   // Retrieval info: PRIVATE: Clock_A NUMERIC "0"
139   // Retrieval info: PRIVATE: Clock_B NUMERIC "0"
140   // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
141   // Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
142   // Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
143   // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
144   // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
145   // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
146   // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
147   // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
148   // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
149   // Retrieval info: PRIVATE: MEMSIZE NUMERIC "128"
150   // Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
151   // Retrieval info: PRIVATE: MIFfilename STRING "ram16x8.mif"
152   // Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
```

```
153   // Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
154   // Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "1"
155   // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
156   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
157   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
158   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
159   // Retrieval info: PRIVATE: REGdata NUMERIC "1"
160   // Retrieval info: PRIVATE: REGq NUMERIC "1"
161   // Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
162   // Retrieval info: PRIVATE: REGrren NUMERIC "1"
163   // Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
164   // Retrieval info: PRIVATE: REGwren NUMERIC "1"
165   // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
166   // Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
167   // Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
168   // Retrieval info: PRIVATE: VarWidth NUMERIC "0"
169   // Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "8"
170   // Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "8"
171   // Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "8"
172   // Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "8"
173   // Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
174   // Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "0"
175   // Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
176   // Retrieval info: PRIVATE: enable NUMERIC "0"
177   // Retrieval info: PRIVATE: rden NUMERIC "0"
178   // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
179   // Retrieval info: CONSTANT: ADDRESS_ACLR_B STRING "NONE"
180   // Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
181   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
182   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
183   // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
184   // Retrieval info: CONSTANT: INIT_FILE STRING "ram16x8.mif"
185   // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
186   // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
187   // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "16"
188   // Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "16"
189   // Retrieval info: CONSTANT: OPERATION_MODE STRING "DUAL_PORT"
190   // Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
191   // Retrieval info: CONSTANT: OUTDATA_REG_B STRING "CLOCK0"
192   // Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
193   // Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
194   // Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING "DONT_CARE"
195   // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "4"
196   // Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "4"
197   // Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
198   // Retrieval info: CONSTANT: WIDTH_B NUMERIC "8"
199   // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
200   // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
201   // Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL "data[7..0]"
202   // Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
203   // Retrieval info: USED_PORT: rdaddress 0 0 4 0 INPUT NODEFVAL "rdaddress[3..0]"
204   // Retrieval info: USED_PORT: wraddress 0 0 4 0 INPUT NODEFVAL "wraddress[3..0]"
205   // Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
206   // Retrieval info: CONNECT: @address_a 0 0 4 0 wraddress 0 0 4 0
207   // Retrieval info: CONNECT: @address_b 0 0 4 0 rdaddress 0 0 4 0
208   // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
209   // Retrieval info: CONNECT: @data_a 0 0 8 0 data 0 0 8 0
210   // Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
211   // Retrieval info: CONNECT: q 0 0 8 0 @q_b 0 0 8 0
212   // Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8.v TRUE
213   // Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8.inc FALSE
214   // Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8.cmp FALSE
215   // Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8.bsf FALSE
216   // Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8_inst.v FALSE
217   // Retrieval info: GEN_FILE: TYPE_NORMAL ram16x8_bb.v TRUE
218   // Retrieval info: LIB_FILE: altera_mf
219
```

Date: January 24, 2024 ram16x8.mif Project: FIFO

Addr	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f	ASCII
00	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	.....